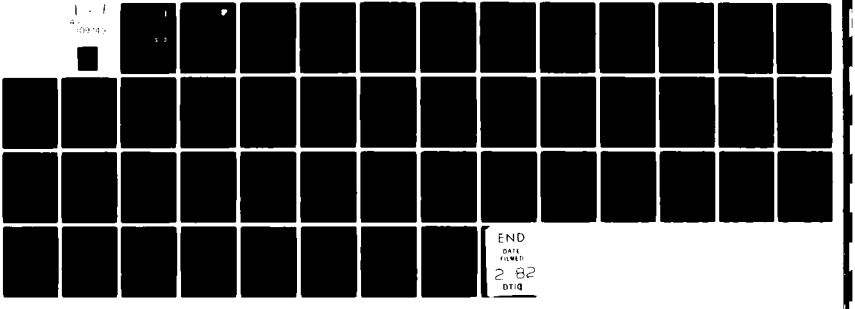
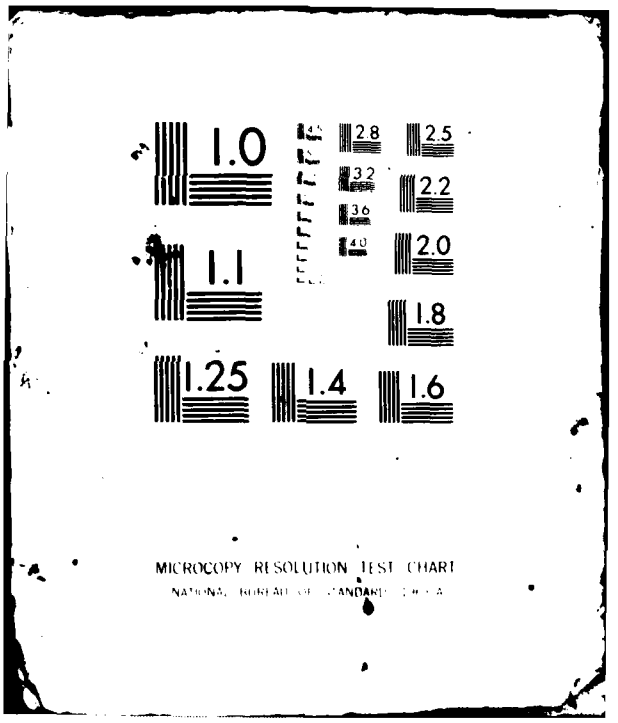
AD-A109 749 COMPUTER SCIENCES CORP FALLS CHURCH VA F/G 9/2
ADA INTEGRATED ENVIRONMENT II SYSTEM SPECIFICATION.(U)
DEC 81 F30602-80-C-0292
UNCLASSIFIED RADC-TR-81-362 NL



END
DATE FILMED
2 82
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MICROCOPY RESOLUTION TEST CHART

PHOTOGRAPH THIS SHEET

AD A109749

DTIC ACCESSION NUMBER

LEVEL

INVENTORY

Computer Sciences Corp
Falls Church, VA

ADA Integrated Environment II
System Specification Dec. 81

DOCUMENT IDENTIFICATION

RADC-TR-81-362

Contract F30602-80-C-0292

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

DISTRIBUTION STATEMENT

ACCESSION FOR
NTIS GRA&I
DTIC TAB
UNANNOUNCED
JUSTIFICATION

BY
DISTRIBUTION /
AVAILABILITY CODES

DIST	AVAIL AND/OR SPECIAL	
A		

DISTRIBUTION STAMP

DTIC
ELECTE
JAN 19 1982
S D
D

DATE ACCESSIONED

82 01 12 005

DATE RECEIVED IN DTIC

PHOTOGRAPH THIS SHEET AND RETURN TO DTIC-DDA-2

DTIC FORM OCT 79 70A

DOCUMENT PROCESSING SHEET

RADC-TR-81-362
Interim Report
December 1981

ADA INTEGRATED ENVIRONMENT II SYSTEM SPECIFICATION

Computer Sciences Corporation

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-81-362 has been reviewed and is approved for publication.

APPROVED: Donald F. Roberts

DONALD F. ROBERTS
Project Engineer

APPROVED:

JOHN J. MARCINIAK, Colonel, USAF
Chief, Command and Control Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COES) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER RADC-TR-81-362	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) ADA INTEGRATED ENVIRONMENT II SYSTEM SPECIFICATION		5. TYPE OF REPORT & PERIOD COVERED Interim Report 15 Sep 80 - 15 Mar 81
		6. PERFORMING ORG. REPORT NUMBER N/A
7. AUTHOR(s)		8. CONTRACT OR GRANT NUMBER(s) F30602-80-C-0292
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Sciences Corporation 803 Broad Street Falls Church VA 22046		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62204F/62702F/33126F 55811918
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (COES) Griffiss AFB NY 13441		12. REPORT DATE December 1981 13. NUMBER OF PAGES 43
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Same		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Donald F. Roberts (COES)
Subcontractors are Software Engineering Associates and Verified Computing Services

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Ada	MAPSE	AIE
Compiler	Kernel	Integrated environment
Database	Debugger	Editor
KAPSE	APSE	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The Ada Integrated Environment (AIE) consists of a set of software tools intended to support design, development and maintenance of embedded computer software. A significant portion of an AIE includes software systems and tools residing and executing on a host computer (or set of computers). This set is known as an Ada Programming Support Environment (APSE). This system specification describes the basic design for a minimal APSE, called a MAPSE. The MAPSE is the foundation upon which an

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

APSE is built and will provide comprehensive support throughout the design, development and maintenance of Ada software. The MAPSE tools described in this specification include an Ada compiler, linker/loader, debugger, editor, and configuration management tools. The kernel (KAPSE) will provide the interfaces (user, host, tool), database support, and facilities for executing Ada programs (runtime support system).

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd)

SECTION 1 - SCOPE

This specification establishes the performance, design, development, and testing requirements for the Ada Integrated Environment (AIE) contract (F30602-80-C-0292) with Rome Air Development Center (RADC). This effort will result in the design, development, and implementation of the Minimal Ada Programming Support Environment (MAPSE), which includes a state-of-the-art Ada Compiler and the support tools and aids necessary for the support of Ada programmers and project managers in the development of Ada software. Procedures for rehosting/retargeting the entire MAPSE (including the compiler and all software development and project management support tools) are required to be developed under this effort.

This specification is prepared in accordance with Military Standard (MIL-STD)-490 and MIL-STD-483 as required by Item A005 of the Contract Data Requirements List (CDRL). When approved and accepted by RADC, it will establish the functional baseline for the MAPSE system.

This submission of the draft system specification is provided for review purposes; final submission will be 30 days after the beginning of Phase II.

SECTION 2 - APPLICABLE DOCUMENTS

The following documents form a part of this specification to the extent specified herein. Additional references contributing to the design are listed in Appendix A.

2.1 GOVERNMENT DOCUMENTS

1. *RADC* Specification No. CP-0787796100E, "Computer Program Development Specification"
2. MIL-STD-483, "*Configuration* Management Practices for Systems, Equipment, Munitions, and Computer Programs"
3. MIL-STD-490, "Specification Practices"
4. Department of Defense (DoD) Requirements for Ada Programming Support Environments, "STONEMAN", February 1980
5. Reference Manual for the Ada Programming Language, July 1980
6. Statement of Work (SOW), Contract No. F30602-80-C-0292, 26 March 1980.

2.2 OTHER DOCUMENTS

7. Computer Program Development Plan (CPDP), developed under Contract No. F30602-80-C-0292.

SECTION 3 - REQUIREMENTS

This section delineates the requirements for the design of a MINIMAL Ada Programming Support Environment (MAPSE) as presented in the SOW for the Ada Integrated Environment (AIE) contract. The system definition, defined in Paragraph 3.1, includes a description of system requirements and major interfaces, including a list of functional areas. System characteristics are presented in Paragraph 3.2. Design and implementation standards to be used in the development of the MAPSE are described in Paragraph 3.3. The MAPSE system documentation that will be produced is detailed in Paragraph 3.4. The logistics requirements are presented in Paragraph 3.5. Personnel and training requirements are stated in Paragraph 3.6. Functional area characteristics for each of the MAPSE components are presented in Paragraph 3.7.

3.1 SYSTEM DEFINITION

This paragraph identifies the functional areas of the MAPSE and the individual components to be developed. A general description of the system components is presented, followed by an illustration of the functional interfaces. All major system interfaces are defined in the following paragraphs. This paragraph includes a list of the Computer Program Configuration Items (CPCIs) that are a part of the MAPSE system configuration. Also contained in this paragraph are the system requirements based on mission, threat, and operational/organizational concepts.

3.1.1 General Description

A Kernel Ada Programming Support Environment (KAPSE) and a set of MAPSE tools comprise the MAPSE system. The KAPSE consists of two CPCIs: the KAPSE Data Base System (KDBS) and the Kapse Framework (KFW). The MAPSE tool set consists of six CPCIs: the APSE Command Language Interpreter (ACLI), the Configuration Management System, the Ada Compiler, the Linker, the Editor, and the Debugger. A general description of the KAPSE/MAPSE system is given in the next paragraph. Details of the functional requirements for each of the CPCIs are stated in Paragraph 3.7.

3.1.1.1 KAPSE System Description

In conformance with the characteristics of an APSE, as contained in Sections 1 and 2 of STONEMAN, the KAPSE must be developed to provide a machine-independent portable interface to a spectrum of Air Force and DoD computer systems. To provide such a machine independent design, the KAPSE must provide a logical data base structure, structures for establishing communications between various components of the MAPSE, well defined host interfaces, as well as run-time support facilities. The design of the KAPSE must provide an abstraction of the operating environment while taking into account the invariancies required by the Ada language.

The KDBS serves as the manager of all system and user generated data that is stored in the KAPSE data base. Facilities provided by the KDBS include data base object manipulation, attribute, version, and access control, and support of Ada Standard Input/Output (I/O).

The KFW provides run-time support facilities, process scheduling, and the host environment interface for the MAPSE. The KFW is responsible for mapping MAPSE system requests into host facilities. This includes internal MAPSE process scheduling and resource management capabilities that are physically carried out by the host.

Interface between the KAPSE and the MAPSE tool set is provided through the KAPSE virtual interface, which consists of Ada specifications for all functions required to invoke system tools or other programs, interact with the KDBS, and communicate with the KFW.

3.1.1.2 MAPSE System Description

As an approach to the development of an APSE, STONEMAN defined a KAPSE surrounded by a minimal tool set to be the formation of a MAPSE. The MAPSE tool set must include an Ada compiler, text editor, debugger, terminal interface routines, project/configuration control function, linkers and loaders (as required), and any other tools defined during the preliminary design phase of this effort. The tools defined under the MAPSE are required to meet the portability goals of rehostability/retargetability with minimal system impact and thus must be supported by the KAPSE functions,

system/subsystem interfaces, and communication mechanisms. All MAPSE tools shall be written in Ada, communicate with each other and with the KAPSE subsystem using a uniform, consistent format that conforms to standard interface specifications.

The MAPSE tools to be included in the AIE are the ACLI, the Configuration Management System, the Ada Compiler, the Linker, the Editor, and the Debugger. A brief description of each of the MAPSE tools follows.

Through the APSE Command Language Integrator (ACLI), the user can access the facilities of the KAPSE or invoke other system tools or user programs. MAPSE commands are interpreted by the ACLI for appropriate action by other MAPSE components.

The Configuration Management System provides a tool to handle, in a unified manner, version, history, and configuration control.

The Compiler translates Ada source programs into efficient executable code and provides meaningful diagnostic and error messages to the user.

The Linker ensures that separately compiled units are correctly linked into the executable code which the loader loads for execution.

The Editor provides the capability to edit Ada source programs as well as program documentation. It provides line and string insert, move, copy, delete, and minimal word processing facilities.

The symbolic Debugger provides source program references to aid the programmer in debugging his code. The capability for inserting breakpoints and tracing program execution is included.

3.1.2 Mission

For Air Force application, the APSE to be developed shall run on a host machine and support development of software for an embedded target machine. Although retargeting to an embedded target machine is not included in this effort, the Air Force intends to retarget the compiler to a number of machines as soon as it is feasible.

Thus, target code generation and optimization is an important consideration in the design of the Ada compiler. The compiler must not only generate

highly efficient target code; it must also be modifiable to generate code for a multitude of targets. Because of the overall APSE portability requirement, the compiler must be easily rehostable to other host development machines.

Rehostability for the entire MAPSE system is also a requirement of this contract: the initial system will be rehosted once under the Phase II Implementation contract. Therefore, machine-dependent capabilities shall be identified and designed to facilitate rehosting.

3.1.3 Threat

The primary threat involved in the use and operation of the MAPSE is the possibility that MAPSE system or user errors will cause interruption, catastrophic error, or compromise of the whole host system or the MAPSE components. Archive/backup facilities will be provided to maintain integrity of the data base in the case of system/MAPSE failure. This fail-soft capability is enhanced by controlling access to all MAPSE/user programs in order to preserve data within the MAPSE system.

3.1.4 System Diagrams

A schematic of the MAPSE components is given in Figure 3-1. This figure illustrates the division between the MAPSE tools and the KAPSE and shows that certain KDBS and KFW functions are visible at the MAPSE interface level (KDBS Utility Package, KFW Interface Package, and the Ada Run-Time Support (RTS) Package), while other KDBS and KFW functions are available only at a more privileged level. All system tools and user programs access the facilities of the KAPSE through the KAPSE virtual interface, depicted as the area between the MAPSE tools and the KAPSE.

Figure 3-2 illustrates the functional interfaces between the MAPSE components. All system tools and user programs share a common interface to the KAPSE. Within the KAPSE, the KDBS communicates with the host through the KFW, which is the only component that directly accesses the host system facilities. Also indicated in this schematic are the functional components of the KDBS and the KFW, which are to be designed to provide the necessary

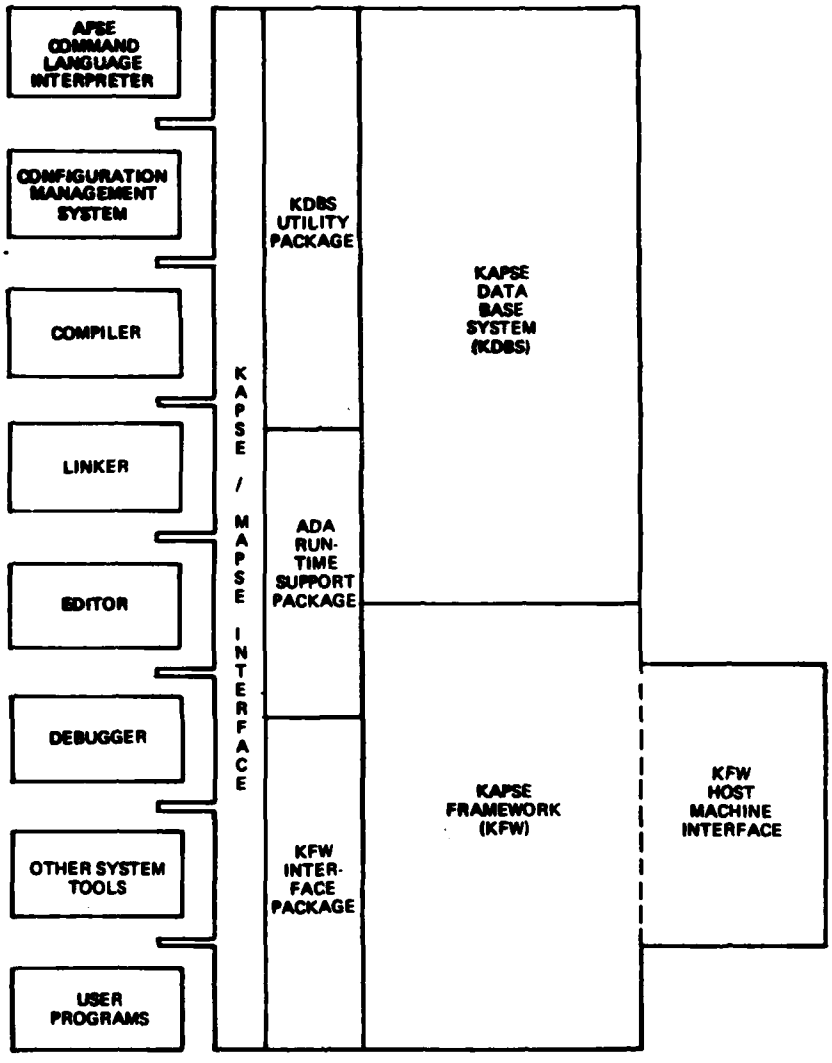


Figure 3-1. MAPSE Component Diagram

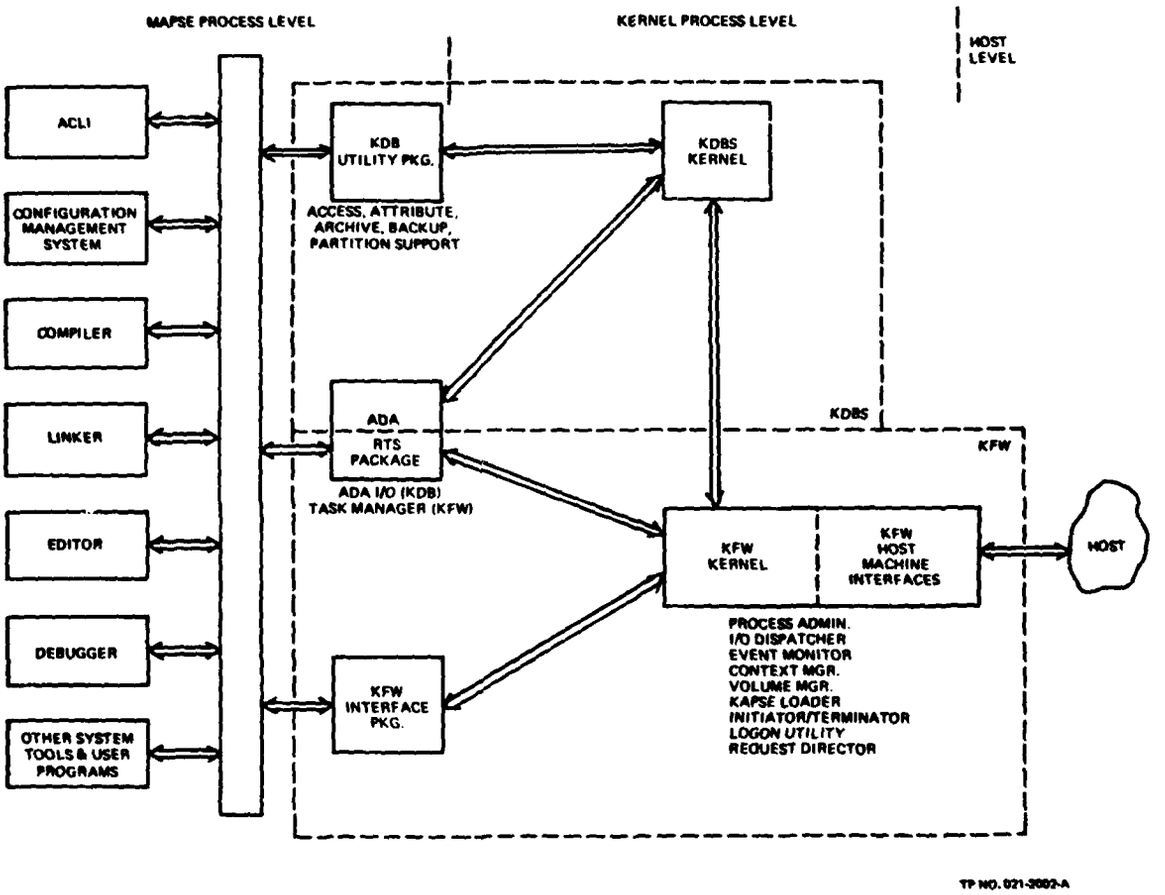


Figure 3-2. MAPSE Interface Diagram

functionality of the MAPSE system. These interfaces are described in more detail in Paragraph 3.1.5.

The system specification tree for the MAPSE design is depicted in Figure 3-3. Each CPCI is indicated on the tree along with the specifications and test documentation required under the contract. The order of development along each branch of the tree is also shown in the figure.

3.1.5 Interface Definition

There are the three major types of interfaces within the MAPSE system: the user interface, through which the user accesses the facilities of the MAPSE; the MAPSE-to-KAPSE interface, through which the MAPSE tools communicate with the KAPSE and with each other; and the interface between the KAPSE and the host environment, enabling the mapping of KAPSE facilities into host facilities. All KAPSE external interfaces (host, user, tool) shall be isolated, clearly identified and designed to facilitate portability of the KAPSE and any MAPSE tool. Uniform protocol conventions shall be specified for communication between users, tools and the KAPSE/MAPSE. Format for invoking and using any of the KAPSE/MAPSE facilities shall be uniform and/or identical wherever possible.

Requirements for each of the major interfaces, derived from the SOW and requirements analysis, are discussed in the following paragraphs.

3.1.5.1 User Interface

The APSE Command Language Interpreter (ACLI) is the medium through which the user accesses and interacts with the MAPSE. The ACLI provides the user with the capabilities to initiate, terminate, and communicate with executing programs. Capabilities for users and tools to execute Ada programs shall be provided through the ACLI. User commands for job control and invoking tools shall have uniform formats.

The functionality of the ACLI is made available to users through the APSE Command Language (ACL). Requirements for the syntax and semantics of the ACL are derived from human engineering considerations and are listed below.

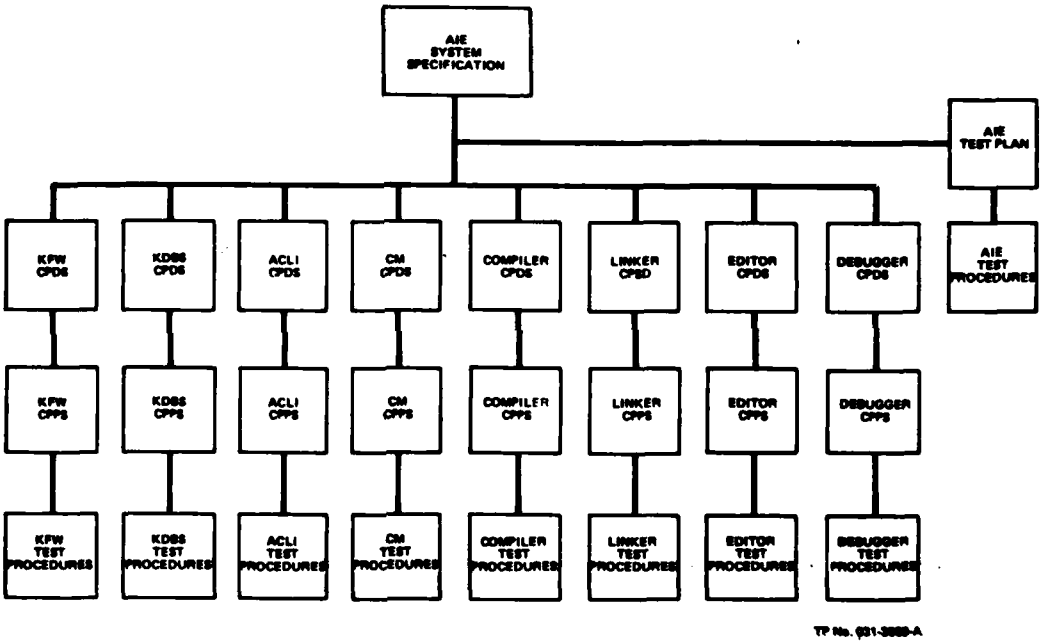


Figure 3-3. MAPSE Specification Tree

The ACL shall:

1. Be derived from a few simple concepts that are straightforward to understand and use
2. Provide a simple mechanism for invoking and communicating with system tools and user programs
3. Provide minimal Ada-like control statements, including case statements, conditional statements, and loop statements
4. Permit the standard input, standard output, and standard error files to be redirected
5. Provide for background execution of programs as well as the pipelined composition of programs
6. Permit the dynamic instantiation in a command of the standard output of an executed program
7. Be essentially free-format: statements may be one per line, several per line, or broken across several lines
8. Provide scalar and composite variables
9. Be expressible in the standard Ada character set, but permit the user to alter or substitute for the interpretation of any special character or control key
10. Include the concept of a user "profile" that specifies terminal characteristics, environment data, and commands that are automatically invoked at logon
11. Permit command language procedure objects to be created and executed.

The user interface shall also include the capability to interface with each of the tools in the MAPSE tool set. These interfaces shall be consistent and follow standard conventions for similar operations.

Through the ACLI the user shall have access to all of the facilities of the KAPSE and the MAPSE tools.

Utilities are to be provided for data base mainpulation, communication with and control of executing Ada programs, and to generate the following reports:

12. Configuration Composition Report - For a given configuration, a report containing the names of objects in the configuration

13. Attribute Report - For a given object, a report containing the attribute names and values associated with the object

14. Partition Report - For a given partition component, a report containing the names of the objects in the component

15. Attribute Select Report - For a given attribute name and value, a report containing all names of objects having that attribute.

16. Summary Reports - Reports based on combinations of attribute, partition, configuration, or version qualifier shall also be provided. For example, a report containing names of all objects having category attribute "Ada source" and contained in Partition "Configuration Item = ABC".

3.1.5.2 MAPSE-to-KAPSE Interface

Host-independent interfaces for the user and MAPSE tools shall be specified. A virtual interface shall be specified for all KAPSE/MAPSE communication.

All tools and user programs depend on the KAPSE virtual interface. Components of the KAPSE (the KDBS and the KFW) are perceived by the MAPSE through this interface. In addition, all interactions between MAPSE processes (both user programs and system tools) are logically handled through the same interface.

The virtual interface shall, through the functions of the ACLI, provide the user the capability to invoke MAPSE tools and to interact and exercise control over the invoked tool. The capability to invoke any MAPSE tool from any other MAPSE tool shall be provided.

To support the design goal of an integrated and open-ended system, shared data (data produced by one MAPSE tool that is needed or useful to another tool) shall be specified and provided as standard interface objects. The interface objects are the medium through which the MAPSE tools communicate. There are six major object format definitions required:

1. Format of a text object (including Ada source object representations)
2. Format of intermediate language objects
3. Format of a compiled compilation unit (including symbol table, source line mapping, program topology, and any other information required by the Linker, Debugger, and Compiler)
4. Format of a fully or partially linked program - note that this may be the same as Item 3 above
5. Format of an executable Ada program
6. Format of Ada Standard I/O objects.

All of the object definitions shall be expressed as abstract data types; that is, as packages defining both data format and access functions. This abstract definition is an absolute requirement because an interface object is likely to have different underlying implementation representations on various machines.

Uniform protocol conventions shall be specified for communication between users, tools and the KAPSE/MAPSE. The formats for invoking and using any of the KAPSE/MAPSE facilities shall be uniform and identical wherever possible. In particular, the following functional interfaces shall be included:

7. Data Base Interface

   a. Open, close, create, and delete objects

   b. Modify object attributes and relationships (access, owner, category, partition and configuration membership, history, version)

   c. Ada Standard I/O

   d. Object access validation

8. Process Communication Interface

   a. Create a process

   b. Signal a process

   c. Terminate a process

   d. Change the priority of a process

   e. Establish a pipe communication channel between processes

9. Other interface requirements

   a. Access *and/or update accounting* data *for* process execution (resource usage)

   b. Initiate execution time profiling

   c. Get and set MAPSE system time

   d. Get and set MAPSE system resource limits.

Not all of these functions are available to every user. Many system functions will verify process ownership before proceeding with requests that could compromise system integrity and security.

3.1.5.3 MAPSE-to-Host Interface

The KAPSE must provide standard interface specifications and functions to support communications between MAPSE users and host facilities. The user will access the system to invoke a MAPSE tool or to use KAPSE facilities to *execute an Ada program.* The KAPSE shall provide the basic initiation and termination (logon/logoff) facilities and standard interfaces to run-time

support and host facilities required to perform the desired function. Therefore, the KAPSE shall provide job control facilities, including interfaces to required host facilities and functions to interpret and execute job commands.

The KFW shall be designed to supply this functionality to the MAPSE. The KFW defines a virtual operating system that provides a canonical interface to the host environment in order to control, execute and support the portable MAPSE components. This virtual operating system shall present a consistent, uniform, machine-independent interface to the rest of the MAPSE system and shall interface directly with the host facilities to provide the functionality of the underlying operating system.

Standard terminal interface specifications and functions shall be provided to facilitate use of a variety of batch and interactive terminals. The specifications shall include protocols for asynchronous user interactions and provide standards for implementing simple editing of a command line.

Host interfaces required to support the low level KAPSE I/O function and the high level I/O package shall be specified.

MAPSE interfaces to the host are required for the following facilities:

1. Startup processing
2. Console control
3. Clock and time information
4. File management
5. Peripheral device control
6. Process control
7. Interrupt handling
8. Accounting information.

3.1.6 Government-Furnished Property List

The Government shall furnish the computers on which the MAPSE will execute. These are: IBM VM/370 and the Interdata 8/32 (OS/32).

Both machines will be located at RADC, Griffiss Air Force Base. The computers will be made available for final system testing 2 months prior to the completion of Contract Line Item Number (CLIN) 0005, 0006, and 0007 to ensure that sufficient time is provided for the orderly transition of the software.

3.1.7 Operational and Organizational Concepts

The AIE, after delivery, can be made available to other Air Force sites possessing the requisite host computers. With modification, the system can be deployed to all Air Force sites where software development is a significant activity.

3.2 CHARACTERISTICS

This section identifies the performance, reliability, and maintainability requirements for the MAPSE.

3.2.1 Performance Characteristics

Reasonable performance, which is usually measured in terms of response time for interactive systems and turnaround time for batch systems, is a prime requirement of the MAPSE. The compiler will compile at the rate of 1000 lines per minute or more. Specific timing requirements for other MAPSE components are dependent on the host configuration and the number of users.

3.2.2 Physical Characteristics

Not applicable.

3.2.3 Reliability

Reliability shall be provided by designing the MAPSE to be tolerant of user and system errors. Archiving and backup/recovery capabilities shall be provided in the KDBS to enhance data base integrity. Suitable warning and error messages shall be generated by the compiler to ensure that user programs do not adversely affect system operation. Attention to the design goals of user-friendliness and simplicity of use throughout the MAPSE aids in preventing many user-generated errors.

3.2.4 Maintainability

Maintainability of the MAPSE is enhanced by the fact that all components are designed with a consistent philosophy. Isolation of machine-dependent features allows the MAPSE to accomodate maintenance and upgrade of host systems. Standard visible specification of facilities provided by the MAPSE also supports maintainability.

3.2.5 Availability

The MAPSE shall be designed to provide a high rate of availability. This will be accomplished through sufficient fail-soft mechanisms as well as providing adequate documentation and training to enable Government personnel to keep the system functional.

3.2.6 System Effectiveness Models

Not applicable.

3.2.7 Environmental Conditions

Environmental factors will not affect the MAPSE except insofar as they affect the operation of the underlying hardware.

3.2.8 Nuclear Control Requirements

Not applicable.

3.2.9 Transportability

Not applicable.

3.3 DESIGN AND IMPLEMENTATION STANDARDS

3.3.1 Materials, Processes, and Parts

Not applicable.

3.3.2 Electromagnetic Radiation

Not applicable.

3.3.3 Nameplates and Product Marking

Not applicable.

3.3.4 Workmanship

Not applicable.

3.3.5 Interchangeability

Not applicable.

3.3.6 Safety

Not applicable.

3.3.7 Human Performance/Human Engineering

The MAPSE shall be designed with human performance/engineering concepts in mind. These concepts include tolerance of user-generated errors, consistency across host environments, ease of use, and provision of adequate training and maintenance materials.

The MAPSE system shall be tolerant of user-generated errors. The system will not be rendered inoperable by a false keystroke on the part of a user. The capability of editing commands interactively shall be provided. Archiving and backup facilities shall be provided to maintain the integrity of the user data base.

A consistent user interface that will not change from implementation to implementation shall be provided. This ensures that MAPSE users will use the system in the same manner on any host supporting the MAPSE.

Requirements for ease of use with a minimum of training shall be incorporated into the design. The user command language will contain simple, meaningful commands. Interfaces to each of the tools in the MAPSE tool set shall be consistent and follow standard conventions for similar operations.

Training courses for the operation, use, maintenance, rehosting, and retargeting of the MAPSE shall be provided. Sufficient design documentation, as well as user-oriented documentation shall be delivered at the end of Phase II. Training is addressed in Paragraph 3.6 and system documentation is listed in Paragraph 3.4.

3.3.8 Computer Programming

The MAPSE system shall be designed and implemented in accordance with the standards documented in the CPDP. Design and implementation standards to be used for the MAPSE are in compliance with RADC Specification No. CPO787796100E.

Design standards include a top-down hierarchical approach, using phased development to provide early functionality. Program modules shall not be released to be coded prior to completion of detailed design. Design walkthroughs shall be used to verify that the design is complete and consistent. A program design language (PDL) shall be used to express the design. The PDL for each module shall include a prologue, which detail the functionality of the module, the internal and external interfaces required as well as listing originator/modification authors and the corresponding dates. The PDL shall contain descriptions of variables, types, constants, exceptions, and large data structures used by the module. The PDL shall also provide a description of the function to be performed by the module.

Implementation standards shall include requirements for coding, unit testing, and code change control. Every program shall be required to contain a prologue and the actual code. The prologue shall follow the format and standards for the PDL prologue. The code shall be written in Ada and standards for naming conventions, data declarations, code indentation, string handling, and condition handling shall be followed. Formal in-line annotations for readability and maintainability shall be required. Unit testing shall be accomplished by the developers and reviewed for sufficiency by independent testers. When the code has been successfully unit tested, it shall be placed under project configuration control. Any subsequent changes to the code must be approved before a new version is made available.

3.4 DOCUMENTATION

The plan for MAPSE system documentation is detailed in the CPDP. The system and user-oriented documents that will be produced during the implementation phase are:

1. System Specification - Final version of the Type A specification for the MAPSE

2. Computer Program Development Specifications - Final versions of the Type B5 specifications for each of the MAPSE components

3. Computer Program Product Specifications - Type C5 specifications for each of the MAPSE components

4. Computer Program Listings - Submitted on magnetic tapes and as listings

5. Maintenance Manual - Information necessary to effectively maintain the MAPSE system

6. Users Manual - Information necessary to effectively use all of the MAPSE facilities in both interactive and batch modes

7. Language Reference Handbook - Reference handbook describing the syntax of Ada, the ACL, and other tool-related command languages

8. Retargetability/Rehostability Manual - Instructions necessary to effectively rehost/retarget the MAPSE system to another computer system, describing in detail the procedures necessary to add or delete various tools.

Descriptions of the above documentation as it relates to the MAPSE CPCIs is presented in the corresponding B5 specifications. All required documents shall be prepared in accordance with the applicable Data Item Description (DID).

3.5 LOGISTICS

This paragraph, defines the logistics requirements on the maintenance, supply and facilities requirements of the AIE.

3.5.1 Maintenance

Maintenance support for the KAPSE/MAPSE, on both the IBM VM/370 operating system and the Interdata 8/32 under the OS/32 operating system, will be supplied for a period of 12 months following delivery and acceptance of the KAPSE/MAPSE system.

3.5.2 Supply

During the maintenance phase, error correction in software as well as requisite documentation updates shall be performed by Computer Sciences Corporation (CSC). In addition, any modification to the system emerging from changes in the Ada language specifications or changes required to meet specific Air Force requirements will be incorporated by CSC. Software or system changes will be performed on both the IBM 370 and Interdata 8/32 systems and appropriate updates to system magnetic tapes and source listings as well as applicable user documentation will be delivered in accordance with defined contract schedules.

3.5.3 Facilities and Facilities Equipments

The MAPSE system shall be developed on CSC computer facilities but shall be delivered and installed on the Government-furnished IBM VM/370 and Interdata 8/32 with OS/32 operating system at RADC.

3.6 PERSONNEL AND TRAINING

This section addresses personnel and training requirements for the use, operation, and maintenance of the MAPSE.

3.6.1 Personnel

After the MAPSE system has been installed at RADC, it will operate as a facility supplied by the IBM and Interdata machines. The number of operators is limited by the hardware available. Initial maintenance, for the first year after delivery, shall be supplied by CSC. Grades and number of personnel required for follow-on maintenance is to be determined.

3.6.2 Training

Training courses shall be provided for Government personnel in the use, operation, maintenance, rehosting, and retargeting of the MAPSE. The users course is to be a one-week course in the use and operation of the MAPSE. The two-week maintenance course provides instruction in the maintenance,

rehosting, and retargeting. Adequate training materials, a User's Manual, a Maintenance Manual, and a Rehostability/Retargetability Manual shall be delivered to support the initial presentations and the ongoing training of RADC personnel. The training location will be at RADC.

3.7 FUNCTIONAL AREA CHARACTERISTICS

This section describes the characteristics of each of the functional components of the MAPSE identified in Section 3.1. The MAPSE system is composed of six CPCIs. The KAPSE CPCIs are the KFW and the KDBS. The MAPSE tool set CPCIs are the ACLI, the Configuration Management System (CMS), the Compiler, the Linker, the Editor, and the Debugger. The functional requirements for each of these CPCIs, as determined by the SOW and our requirements analysis, are discussed in the following paragraphs.

The basis for the design of the AIE is the KAPSE/MAPSE approach as introduced in STONEMAN and detailed in the SOW. General requirements for the design of the KAPSE/MAPSE are stated below. Subsequent paragraphs detail the requirements for the individual components of the system.

The KAPSE/MAPSE shall be portable to the maximum extent possible.

The KAPSE/MAPSE system design shall include features to protect itself from user and system errors.

KAPSE/MAPSE software shall be designed to be modular and reusable. Software performing a single function required (or potentially required) by more than one system component shall be designed to be reusable to the maximum extent possible. The method of achieving the reusability (such as subroutine, functions, package, in-line section of code, etc.) shall be decided based on the function and other requirements for the components in which it is to be used. In any case, KAPSE/MAPSE software shall be designed to incorporate software reusability and documented to encourage it.

3.7.1 KAPSE System Characteristics

Functional requirements for each of the KAPSE components are delineated in the following paragraphs. The general requirements for the KAPSE system listed here shall be incorporated in the design of the components.

The KAPSE shall provide all of the data base support, interfaces to host hardware and software facilities, user interfaces, and tool interfaces to satisfy user and tool requirements.

The KAPSE shall provide basic run-time support facilities that are required to execute Ada programs.

3.7.1.1 KAPSE Framework (KFW)

The KFW shall provide the operating system facilities for the MAPSE and run-time support for Ada programs. It shall ensure portability of the data base and perform control functions necessary for the MAPSE to effectively execute either on a bare machine or in cooperation with an underlying host operating system.

Basic data transfer and control functions required to support the high level I/O package shall be provided.

The KFW shall support:

1. A KAPSE/MAPSE initiation and termination function.
2. Process administrative functions required to execute Ada processes and meet the portability goals as defined elsewhere in this specification. This shall include the facility for loading Ada programs.
3. Task management functions to provide Ada tasking as required by the implementation of full Ada facilities.
4. Memory management functions as required by a rehostable/ retargetable design.
5. Event monitoring.
6. Facility requests for host-dependent functions.
7. Requests for host-dependent file management facilities.

The KFW shall support and coordinate requests from concurrent processes resulting from data base manipulation or from low level I/O.

The KFW shall support a high level I/O package as an extention of, or alternative to, the package specified in the Ada Reference Manual. The package shall provide capabilities comparable to facilities typical in existing Formula Translation (FORTRAN) systems.

The KFW shall support terminal interface routines for batch and online keyboard terminals required for Phase II implementation. The specific terminals selected shall be based on the computer facilities to be used for development.

The KFW design shall identify, and where possible specify, any additional host dependent computer programs necessary to implement the KAPSE/MAPSE system on the IBM VM/370 and Interdata 8/32 computers specified for delivery of the system in Phase II.

3.7.1.2 KAPSE Data Base System (KDBS)

The KDBS is the central feature of the MAPSE. It shall provide the repository for all information on a project throughout the project life cycle. All software development data shall be represented as objects, attributes, and relationships in a data base managed by an information management system. The KDBS shall be designed to facilitate access to object attributes and relations for the purposes of report generation.

The primary constituent of the KAPSE data base is an object, the fundamental, elementary entity that the KDBS manages. An object has a unique name, contains information, and has attributes. The KDBS shall provide the capability to create, delete, modify, store, retrieve, input, and output data base objects. Additional operations or alternate forms of the basic operations, such as facilities for version control, configuration definition and storage, and attribute manipulation, shall also be specified and provided in the design as required. The object management facilities shall be virtualized to provide portability.

The KDBS shall provide for all forms of data necessary to fulfill all of the general requirements for the AIE. The content and form of the data base shall be consistent with these requirements.

The KDBS shall not be dependent upon any external software systems (such as data base management systems) that are not specifically an integral part of the operating system of the host computer.

The KDBS shall support the creation and storage of Ada libraries in source form.

The KDBS shall provide the capability to define new categories of objects (other than those specified by this effort) without imposing restrictions on the computer information stored in those objects.

The KDBS shall provide flexible storage facilities to all MAPSE tools. It shall provide the capability to read and write data base objects from within any MAPSE tool, using the facilities required to support basic runtime facilities and high level I/O. The MAPSE tools shall use the KAPSE data base as an information repository and exchange. One tool creates an object, saves information in it, and then invokes another tool, which, in turn, uses that information to perform additional actions for its caller. This is a method of intertool communication that is flexible and independent of special operating system mechanisms.

The KDBS shall provide the capability to create partitions as defined in the following paragraphs.

For management purposes, it is often desirable to impose logical structures on project software. Such structures are called "partitions" in this document. For example, a project manager must be able to designate levels of development for migration of software during the development process (e.g., code, debug, unit test, integration test, operational). Strict rules and access controls must be imposed for migration of software from one level to the next. Management reports based on the established structure shall be provided. Another example of a partition is the structuring of a software by function (for example, tracking, navigation, data management). Note that both of these illustrative partitions may be imposed on the same software project. The rules for forming a partition are not generally known by the

system in advance; hence those rules are not a well-formed configuration group. The KDBS shall accommodate user creation and maintenance of partitions including the facility to dynamically add or remove objects from partitions.

The KDBS shall provide the capability to assign version qualifiers to objects or groups of objects. As a minimum, the capability to assign time/date stamps, and serial number version qualifiers shall be provided. Assignment of the version qualifier shall be at the option of the user/project manager and not a prerequisite for objects. The capability to designate and use a default version shall also be provided.

The KDBS shall provide a general capability to create object attributes. The following predefined attributes shall be provided: history, category, and access.

The history attribute records the manner in which the object was produced and all information related to the production of the object. The history attribute shall contain sufficient information to provide a basis for comprehensive configuration control. Necessary constraints shall be imposed on data base operations so that the validity and consistency of history attributes is ensured.

The category attribute indicates the category of information contained in the object (source, documentation, relocatable, ...). It shall be possible for the category attribute to be used in such a way that MAPSE tools are offered protection against accessing an object in a way that it is not meaningful (incompatible with the format or content of the object) but are not prevented from accessing an object in any way that is meaningful.

The access attribute indicates access rights to the object. The data base system shall provide capability for controlling access to data base objects based on version qualifier, attributes, and partitions. The access controls shall be programmable in the sense that controls may be modified or redefined from project to project. A privileged user (such as the project manager) shall be able to override user-defined access controls in order to establish project control. Access controls shall be specified as a design task.

The KDBS shall provide the capability of archiving data base objects. The archiving process shall retain the integrity, consistency and eventual availability of data base objects. The KDBS shall also provide for backup and recovery of a user's data base objects.

The visible interface to the KDBS shall be defined as Ada package specifications within the KAPSE virtual interface.

3.7.2 MAPSE Tool Set

The following requirements shall be incorporated in the design of the MAPSE tool set.

Intertool communication shall be through the virtual interface facilities.

To support design goals for an integrated and open-ended system, data produced by one MAPSE tool and needed or useful to another tool shall be saved. One example of such data is the intermediate language representation of an Ada program. Interface specifications for such data shall be provided.

MAPSE tool designs must satisfy requirements for portability, modularity, flexibility and ease of use.

All MAPSE tools shall be developed in Ada.

Support for the Ada program libraries as defined in [5] and required to support the compilation and linking of Ada programs shall be provided by the Compiler in conjunction with the KDBS. The design of the Ada program library shall be fully optimized.

The Compiler shall include a mechanism for automatic stub generation. Stubs are normally employed, in top-down design in the testing/integration phases, where a coded higher-level software unit may invoke a lower-level software unit which has not yet been coded. Stubs shall be assigned the name of the uncoded data base unit and shall contain the interface arguments necessary to permit compilation and execution of the higher-level software unit. The KDBS shall store the source code for the generated stub and maintain pertinent information for the generated stub.

3.7.2.1 APSE Command Language Interpreter (ACLI)

The MAPSE shall provide for a simple and efficient user interface. This is embodied in the ACLI.

The ACLI shall initiate, name, and communicate with named processes.

The ACLI shall have controlled access to certain KAPSE data base objects for the validation of passwords and access rights.

Where possible, the ACLI shall be able to modify its instantiation of the terminal handler to provide different interpretations for control keys.

Formats for similar user commands shall be uniform and consistent across all tools.

User/tool communication conventions must be designed to allow the user to communicate naturally and consistently across all tools.

3.7.2.2 Configuration Management System (CMS)

Facilities to support project configuration management shall be provided by the CMS in conjunction with the ACLI. CMS shall process configurations, which define sets of data base objects combined with rules describing how these objects are to be derived from each other. These rules are to be specified in ACL.

CMS processing shall include the following functions:

1. Update - To determine and perform the minimal set of operations that are required to bring specified objects in a configuration up-to-date with respect to the other objects in the configuration.
2. Reconstruct - To determine and perform the minimal set of operations that are required to reconstruct a specified version of an object in a configuration.

3.7.2.3 Ada Compiler

The compiler shall be designed to operate in multiple passes, so as to minimize system demands on the host computers. The compiler shall be designed so that available memory may increase as the size of the program being compiled increases.

The compiler shall be designed for batch, remote batch and online use.

The compiler shall be designed to be easily rehosted and retargeted. The design shall provide for the parameterization of machine-independent parts, and for separable target machine code generator phases. As a design goal, a single compiler front end shall be capable of interfacing with back ends of various target machines. For the most part, these back ends machine dependent characteristics must be kept to a minimum. The interfaces between the front end and various back ends must be clearly defined and documented.

The compiler shall process Ada source programs in the format stipulated by the MAPSE design and produce an efficient equivalent program in the form of an object output unit. Additional compiler inputs shall include unique target machine characteristics, libraries, or other Ada modules required by the source program being compiled.

The compiler shall be designed to process the complete Ada language as specified in the Reference Manual for the Ada Programming Language (July 1980).[5]

Pragmas of the language are used to convey information to the compiler and to issue directives to the compiler. Pragmas for the Ada compiler shall be designed to satisfy requirements contained elsewhere in this document. The language defined pragmas [5] shall also be designed.

The compiler shall produce as output, object code and all other necessary outputs required in order to implement the separate compilation of modules and subsequent linking and execution of programs as specified in the Ada Reference Manual. As a minimum, the compiler shall also produce output listings as stated below, any or all of which can be user supressed. The term "optionally", refers to selection as a user option and not as an option in design.

A symbol attribute listing shall be optionally produced, and shall contain a list of all symbols defined and their characteristics. The listing shall be in the collating sequence defined in [5].

A cross-reference listing shall be optionally produced containing a list of each symbol, its block level and a list containing the statement numbers where the symbol is set and used.

The compiler shall optionally produce a source program output listing contained the statement sequence numbers of all source program input lines processed by the compiler. Each page of this listing shall provide the page number, the date and time the listing was produced, the column numbers of 0 through 80, the program name, and the version of the compiler. Error messages shall appear on the first available line following the source line containing the error. If more than one error is detected per input source line, each error message shall be printed, one error message per line.

The compiler shall optionally produce an object program listing. Each page of the object program listing shall contain the name of the program compiled, time and date of compilation, the compiler version, and the page number. A comment containing the source code statement number and statement shall precede the machine code representation of the statement and an equivalent assembly language representation in "side by side" fashion. This object code listing shall contain all instructions, data, constants, directives, and any other supplemental information generated by the compiler. Adjacent to each object code instruction shall be an equivalent assembly language statement which shall make correct use of all labels as they appear in the Ada source program and shall provide compiler-generated labels where no corresponding source label exists.

The compiler shall optionally collect, store, and output in a suitable format the source program and compilation statistics. The statistics to be collected shall be specified in the B5 Specification.

The environment listing shall contain, ordered alphanumerically in columns, external names referenced or used by the source program compiled (for example, use names, library routines). The listing shall include the source line statement number where the external name is referenced or used.

In addition to the user oriented listings required above, information useful in managing and maintaining the APSE (for example, intermediate language, symbol tables) shall be produced and presented in a readable format.

The compiler shall perform extensive error checking. Errors detected shall be reported and associated with the source line number where the error occurs.

Severities of compiler errors shall indicate the correctness of the resulting object program, and in some cases, inhibit the generation of an object program. When an error is detected, the compiler shall attempt reasonable recovery actions and continue processing to detect as many errors as practicable. However, certain errors may prevent continuation of the compilation process. In these cases, compiler output other than the source listing and error messages shall be inhibited. Severities of compiler errors shall include:

1. Note (N): Information to the user about the compilation process for the source intput program. The compilation process continues and the object program is not affected.

2. Warning (W): Information about the input program having to do with the reliability or validity of the source program or algorithm. The source input program is correct and the translation to the object program well defined. The object program contains code for the questionable construct and may behave incorrectly at run time.

3. Error (E): Identification of an illegal syntactic or semantic construct with a well defined recovery action, possibly in terms of a meaningful linguistic or implementation dependent default. The compilation process continues and the object program (if generated) contains code for the illegal construct. The object program may behave incorrectly or meaninglessly at run time.

4. Serious error (S): Identification of an illegal construct with no well defined recovery action. The analysis portion of the compilation process continues after a reasonable syntax recovery action but no object program is generated. Other outputs, such as environment listing and set-used listing, may be invalid.

5. Fatal Error (F): Identification of an illegal construct or unrecoverable host environment error with no reasonable syntactic recovery action. The compilation process terminates at the point of the fatal error, and no outputs other than the source listing and error messages, including the fatal error, are produced. In any fatal, abortive situation where the compiler can exit properly before job termination, all files opened and accessed during compilation shall be properly closed.

Error messages shall contain an error identifier, severity code, and a descriptive text indicating the precise meaning of the error. Additional information shall, in most cases, be provided by identifying more precisely the cause of the error (for example, by indicating an inappropriately used name).

The compilers shall detect all syntax errors and all semantic errors as identified by language constructs in [5]. The syntactic and semantic errors shall be described clearly. The compiler shall also indicate any capacity requirement that has been exceeded during processing. The severity of such capacity errors shall not cause the compilers to stop processing a source program unless the error has previously been identified as fatal as discussed above. A complete list of all error messages capable of being generated by all compilers shall appear in the Users Manual. For each message listed in the manual, at least one descriptive example of the type of coding error that can cause that message to be produced shall appear under the error message text. An additional list of unique messages issued by the compiler for each target computer shall also appear in the Users Manual for each compiler.

Optimization, the transformation of the program into another representation, without changing the intended effect of the program, for the purpose of reducing the programs resource requirements (such as time or space) will be supported. Any program that compiles successfully without optimization and executes without error shall provide the same result (except for performance) with optimization. Optimization shall occur at the user's

option through language pragmas. Optimization with respect to memory usage and execution speed shall be provided. When memory usage and execution speed are conflicting requirements, they shall be provided for separately. Optimization techniques for the compiler shall be specified.

The following additional features shall be included in the design of the compiler:

1. Source Reformatting
2. Statistics Collection
3. Program Flow Description
4. Language Modularity
5. Retargetability
6. Listing Aids
7. Environment Simulation Support
8. Object Version Genealogy
9. Compilation Order Validation Information
10. System Usage List
11. Symbolic Debugging Facilities.

3.7.2.4 Linker

Facilities for the linking and loading of Ada programs shall be provided. These facilities shall adhere to the rules and specifications relating to linking and loading functions contained in the language manual [5].

The MAPSE Linker is the tool used to collect program units together for loading. It shall accept commands that identify the units to be combined and their allocation. The Linker shall perform symbol resolution and relocation, and produce a new relocatable program.

The Linker shall utilize the Ada Program Library and user commands to locate compilation units and to resolve references.

The design of the Linker and the KFW loader facility shall provide a smooth flow of executable object modules to executing Ada programs.

Along with those requirements listed in the SOW, the following additional features are included.

The Linker shall permit the structuring of programs into multilevel overlays, segmenting of programs into multiple location counters, resolution of external references, specification of symbolic equivalencies and name definition, allocation of stack and heap space, resolution of address references, and evaluation of address equations.

The Linker shall support the MAPSE system in maintaining the history attributes of the object program produced. In addition to the information found in the input programs, the Linker shall include the list of the programs linked to create the new program and the version of the Linker being used.

The Linker shall have the ultimate responsibility to inform the user of compilation order violations and possible interface conflicts.

The Linker shall produce user-oriented listings describing the allocation of the various program location counters and entry points. Included in this listing are the attributes of various location counters such as read-only, read-write, sharable, and self-relocating code. The user may also select a cross-reference listing of the external references.

3.7.2.5 Editor

Basic editing facilities suitable for editing general text such as source programs or program documentation shall be provided in the MAPSE design. The MAPSE Editor shall provide text object creation, modification, and browsing facilities. The following capabilities shall be provided: find, alter, insert, delete, input, output, move, copy and substitute.

The Editor language shall be simple and concise.

Along with those requirements listed in the SOW, the following additional features are included.

The Editor language shall support insertion, replacement, deletion, copying and moving operations on a character-string basis without regard to line boundaries, or on a whole-line basis as requested by the user. Searches are described with optional context.

The Editor shall provide command macro facilities, and the ability to execute conditionally or repetitively a set of commands.

The Editor shall permit the user to invoke the ACLI without exiting the Editor. Any such request nests the Editor process until the request has been satisfied. Provision shall be made to save the results or output from such nested commands for editing and inclusion in the object being created.

3.7.2.6 Debugger

Debugging facilities to assist batch and online users in detecting, locating and correcting errors in Ada programs shall be specified and provided in the MAPSE design.

The debugging facilities shall function at the Ada level in that user input to the debugger and debugger output shall reference Ada source program statements, symbols, variable names, etc.

The debugging facilities shall support debugging of all Ada language features, including concurrent programs.

The debugging facilities shall provide linkage between an executing Ada program in binary form and the corresponding Ada source program.

As a minimum, the debugging facilities shall provide the following capabilities to batch and online users as appropriate:

1. Breakpoints (conditional, preset, dynamically set, single step)
2. Display values (variables and constants)
3. Modify values (variable and constants)
4. Display and modifications of variables in machine representation (such as hexidecimal, octal, or binary) or scalar type representation (such as character, integer, etc.) at the user's option

5. Display subprogram arguments (values and formal names)

6. Modify flow of program (e.g., jump)

7. Tracking

8. Dumps.

The following are additional functional requirements for the MAPSE Debugger.

The Debugger shall provide support for recording path entrance counts and timing statistics to facilitate program tuning and path entrance verification. The user may request program status reports; this includes execution accounting information, where available, current program counter and special machine register values.

In addition to command execution, the Debugger shall permit the implantation of any Debugger command for processing during program execution.

SECTION 4 - QUALITY ASSURANCE PROVISIONS

This section establishes the requirements for formal tests and verifications of system performance and design characteristics and operability. The tests and verifications specified will include subprogram, program (CPCI), system integration and acceptance testing.

Detailed information concerning other project-related aspects of Quality Assurance is provided in the AIE CPDP.

4.1 GENERAL TESTING REQUIREMENTS

The following set of test documentation for each level of testing is provided:

1. Test Plan - Defines the scope of tests required to ensure that the system, function, or program meets all applicable technical, operational, and performance specifications. A draft Computer Program Test Plan will be sumbitted at the end of Phase I and the final plan is required by CDRL item 006 for Phase II submission.

2. Test Procedures - Present detailed instructions for test execution and for evaluation of the results of each specified level of testing. Computer Program Test Procedures are required by CDRL item 007 for Phase II submission.

3. Test Reports - Document the results of a test. They describe, define, and evaluate discrepancies between the program design and the program as produced. As required in CDRL item 007, test records and problem documentation records are to be delivered 10 days after completion of each test.

4.2 RESPONSIBILITY FOR TESTS

Subprogram testing shall be performed to verify that the software interfaces within each CPCI are operating as specified. Program testing shall verify that the CPCI meets the design and performance requirements specified in the

corresponding B5 Specification. System performance testing shall verify that the CPCI satisfies performance and system requirements in the context of the entire MAPSE system. Acceptance testing will be perfomed at the CPCI and system levels. The Computer will be certified using the DoD Ada Complier Validation Facility (ACVF) as part of the acceptance test.

Testing requirements are specified in the B5 Specification for each CPCI. The Computer Program Test Plan, Test Procedures and Test Results will fully document all testing procedures.

Programmers on the MAPSE implementation team shall be responsible for unit testing their respective programs. All testing following the unit testing stage shall be performed by a test team comprising staff members independent of the code developers. Reports describing the test results shall be prepared in accordance with the Government approved CPDP and the Computer Program Test Plan. All formal testing shall be witnessed by the Government.

All testing except acceptance testing shall be done at the CSC facility in Falls Church, Va. Acceptance testing shall be performed at the Rome Air Development Center.

A systematic approach will be employed by the MAPSE program QA staff to ensure that all requirements have been satisfied.

A compliance matrix shall be required to map each test procedure against the system requirements being tested. All requirements, identified by paragraph number and module name, are listed in the compliance matrix in numerical order. As test procedures are developed for subprogram, program, and system testing, they are identified by number and listed in the appropriate row and column in the matrix. One test procedure may satisfy more than one requirement. When testing is completed, all requirements shall have a test procedure listed for at least one level of testing. Many requirements may have test procedures for all levels.

SECTION 5 - PREPARATION FOR DELIVERY

All deliverables produced under this contract shall be in accordance with the CDRL requirements. Magnetic tapes and source listings for all software developed under this contract shall be delivered at the end of Phase II. Documentation shall comply with the associated DID. User documentation (Users Manual, Maintenance Manual, Retargetability/Rehostability Manual) shall be prepared in loose-leaf form to facilitate updates and modifications.

APPENDIX A - REFERENCES

DoD Requirements for the Programming Environment for the Common High Order Language, "PEBBLEMAN Revised", January 1979

AIDA, An Informal Introduction, University of Karlsruhe, 25.11.80.

Design Issues for Ada Program Support Environments, developed for DARPA under Contract No. MDA-903-80-C-0188, October 1980

Formal Definition of the Ada Programming Language, November 1980

Ada Compiler Validation Implementer's Guide, Report 1067-2.3, developed by SofTech for DARPA under Contract No. MDA 903-79-C-0687, October 1980

Ada Support System Study Phases 1, 2, 3, & 4, developed for United Kingdom Ministry of Defence

Ada-M, An Ada-based Medium-level Language for Multi-processing, developed for DARPA under Contract No. MDA 903-80-C-0159, December 1980

ANNA, A language for annotating Ada programs, developed for DARPA under Contract No. MDA-903-80-C-0159

$TCOL_{Ada}$: Revised Report on An Intermediate Representation for the Preliminary Ada Language, developed by the Department of Computer Science, Carnegie-Mellon University, Report No. CMU-CS-80-105, February 1980

Use of Ada for the Design and Implementation of Part of Gandalf, developed by the Department of Computer Science, Carnegie-Mellon University, Report No. CMU-CS-79-135, June 1979

Formal SEMANOL Specification of Ada, developed for Rome Air Development Center, Report No. RADC-TR-80-293, September 1980

Communication Software Development Package System/Subsystem Specification, developed by Computer Sciences Corporation for Rome Air Development Center, Contract No. F30602-79-C-0051, July 1980.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.